4. The Basic Proof Method

From **"**Verification, Validation, and Evaluation of Expert Systems, Volume I"

This chapter provides an overview of the basic method for formal proofs:

- Partition larger systems into small systems.
- Prove correctness on small systems by non-recursive means.
- Prove that the correctness of all these subsystems implies the correctness of the entire system.

Introduction

An expert system is correct when it is complete, consistent, and satisfies the requirements that express expert knowledge about how the system should behave.

For real-world knowledge bases containing hundreds of rules, however, these aspects of correctness are hard to establish. There may be millions of distinct computational paths through an expert system, and each must be dealt with through testing or formal proof to establish correctness.

To reduce the size of the tests and proofs, one useful approach for some knowledge bases is to *partition* them into two or more interrelated knowledge bases. In this way the VV&E problem can be minimized.

Overview of Proofs Using Partitions

The basic method of proving each of these aspects of correctness is basically the same. If the system is small, a technique designed for proving correctness of small systems should be used. If the system is large, a technique for partitioning the expert system must be applied and the required conditions for applying the partition to the system as a whole should be proven. In addition the correctness of any subsystem required by the partition must be ensured. Once this has been accomplished this basic proof method should be applied recursively to the subexpert systems.

To carry out a partitioning of an expert system, one generally requires expert knowledge to define the top level problem-solving strategy of the expert system. In Chapter 7, "Knowledge Modeling", a number of knowledge representations are outlined that may be useful in formalizing the top level structure of the knowledge base. Through knowledge acquisition with one or more expert, the top level structure of the knowledge base should be represented in a knowledge model. The correctness of this knowledge model should be validated with other experts or with standard reference materials in the target domain (the section in Chapter 9, on Validating the Semantic Consistency of Underlying Knowledge Items, addresses the problem of validating expert knowledge). When the formalization of the top level knowledge base has been so validated, the fact that the knowledge base has the validated structure can, from the standpoint of a formal proof, be assumed.

Once the top level structure of the knowledge base has been validated, to show the correctness of the expert system, the following criteria must be accomplished:

- Show that the knowledge base and inference engine implement the top level structure.
- Prove any required relationships among sub-expert systems or parts of the top level knowledge representation.
- Prove any required properties of the sub-knowledge bases.

Chapter 7, "Knowledge Modeling", discusses what exactly must be proved for various knowledge models and for various aspects of the correctness problem.

A Simple Example

To illustrate the basic proof method, Knowledge Base 1 will be proved correct in Figure 5.1. Although this knowledge base is small enough to verify by inspection, the proof will be carried out in detail to illustrate the proof method.

Knowledge Base 1

Rule 1: If "Risk tolerance" = high
AND "Discretionary income exists" = yes
then investment = stocks.

Rule 2: If "Risk tolerance" = low
OR "Discretionary income exists" = no
then investment = "bank account".

Rule 3: If "Do you buy lottery tickets" = yes
OR "Do you currently own stocks" = yes
then "Risk tolerance" = high.

Rule 4: If "Do you buy lottery tickets" = no
AND "Do you currently own stocks" = no
then "Risk tolerance" = low.

Rule 5: If "Do you own a boat" = yes
OR "Do you own a luxury car" = yes
then "Discretionary income exists" = yes.

Rule 6: If "Do you own a boat" = no
AND "Do you own a luxury car" = no
then "Discretionary income exists" = no.

Figure 4.1: Knowledge Base 1

Step 1 -- Determine Knowledge Base Structure

To prove the correctness of Knowledge Base 1 (KB1), the expert knowledge can determine that the system represents a 2-step process:

1. Find the values of some important intermediate variables, such as risk tolerance and discretionary income.

2. Use these values to assign a type of investment.

KB1 was built using this knowledge; therefore, it can be partitioned into the following pieces:

- A subsystem to find risk tolerance (part of Step 1).
- A subsystem to find discretionary income (part of Step 1).
- A subsystem to find type of investment given this information (part of Step 2).

To prove the correctness of a multi-step system, it must be proved that Step 1 satisfies the following criteria:

- For each set of inputs, all the outputs required by Step 2 are always produced by Step 1.
- For each set of inputs, all the outputs of Step 1 are single-valued.
- The correct outputs of Step 1 are assigned to each possible set of inputs.

It must also be proved for Step 2 that:

- For each set of inputs and computed Step 1 outputs, Step 2 produces some output.
- For each set of inputs and Step 1 outputs, all the outputs of Step 2 are single-valued.
- The correct outputs of Step 2 are assigned to each possible set of inputs and computed Step 2 outputs.

Step 2 -- Find Knowledge Base Partitions

To find each of the three subsystems of KB1, an iterative procedure can be followed:

1. Start with the variables that are goals for the subsystem, e.g., risk tolerance for the risk tolerance subsystem.

2. Include all the rules that set subsystem variables in their conclusions. For the risk tolerance subsystem, Rules 3 and 4 are included.

3. Include all variables that appeared in rules already in the subsystem and are not goals of another subsystem.

4. For the risk tolerance subsystem, include "Do you buy lottery tickets" and "Do you currently own stocks".

5. Quit if all rules setting subsystem variables are in the subsystem, or else go to Step 2. For the risk tolerance subsystem, there are no more rules to be added.

Figure 4.2 below shows the partitioning of KB1 using this method.

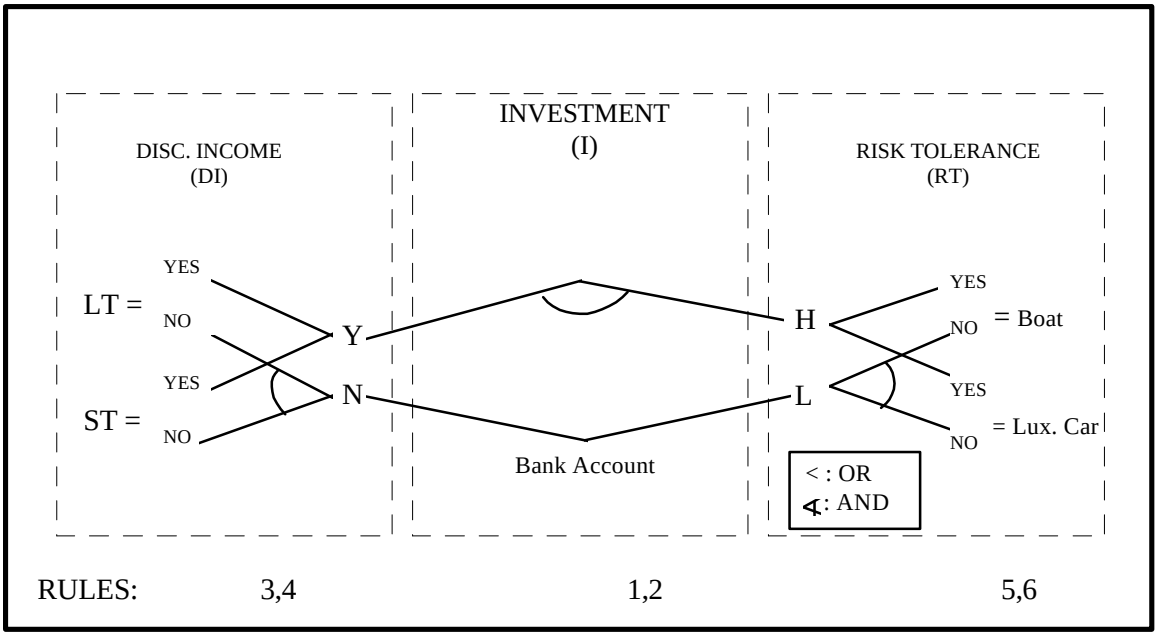

Figure 4.2: An Example of Knowledge Base Partitioning

Step 3 -- Completeness of Expert Systems

Completeness Step 1 -- Completeness of Subsystems

The first step in proving the completeness of the entire expert system is to prove the completeness of each subsystem. To this end it must be shown that for all possible inputs there is an output, i.e., the goal variables of the subsystem are set. This can be done by showing that the OR of the hypotheses of the rules that assign to a goal variable is true.

For example, the discretionary subsystem of KB1 will be shown to be complete. The discretionary subsystem consists of these rules:

Rule 5: If "Do you own a boat" = yes

OR "Do you own a luxury car" = yes

then "Discretionary income exists" = yes.

Rule 6: If "Do you own a boat" = no

AND "Do you own a luxury car" = no

then "Discretionary income exists" = no.

Step 3.1: The first step is to form the OR of the possible outputs of the system:

"Discretionary income exists" = yes (4.1)

OR "Discretionary income exists" = no

(4.1) expresses the condition under which some conclusion is reached.

Step 3.2: For each output condition in (4.1), the user substitutes the OR of rule hypotheses for rules that imply that condition. For example, for

"Discretionary income exists" = yes (4.2)

the only rule inferring (4.2) is Rule 5; its hypothesis is:

"Do you own a boat" = yes (4.3)

OR "Do you own a luxury car" = yes

Since this is the only rule concluding (4.2), (4.3) is the OR of rule hypotheses implying (4.2).

Making the substitution of (4.3) for (4.2) in (4.1), and a similar substitution for:

"Discretionary income exists" = no (4.4)

the result is:

("Do you own a boat" = yes (4.5)

OR "Do you own a luxury car" = yes)

OR

("Do you own a boat" = no

AND "Do you own a luxury car" = no)

Step 3.3: Continue substitutions of the OR of rule hypotheses for inferred propositions (4.5) until the user obtains an expression where only input variables appear. In fact, (4.5) already contains only input variables, and no further substitutions are needed.

Step 3.4: Apply Boolean algebra to simplify the expression from Step 3; the goal is to show that the Step 3 expression always has the truth value TRUE.

Letting:

A = "Do you own a boat" = yes

B = "Do you own a luxury car" = yes

(4.5) can be rewritten as:

(A or B) or (Not A and Not B) (4.6)

Simplifying this gives:

(A or B) or (Not A and Not B)

= (A or B or Not A) and (A or B or Not B)

= true and true

= true

This means that the OR of conditions that imply *some* conclusion is true.

Completeness Step 2 -- Completeness of the Entire System

The results of subsystem completeness are used to establish the completeness of the entire system. The basic argument is to use results on subsystems to prove that successively larger subsystems are complete. At each stage of the proof there are some subsystems known to be complete; initially the subsystem that concludes overall goals of the expert system will be complete. At each stage of the proof, a subsystem that concludes some of the input variables of the currently-proved-complete subsystem is added to the currently complete subsystem. After a number of steps equal to the number of subsystems, the entire system can be shown to be complete.

When a complete subsystem that sets input variables of the currently complete subsystem is added to the latter, the augmented subsystem is complete. Any input to the augmented subsystem can be divided into a set V1 of input variables for the unaugmented system and a set V2 for the newly added subsystem. Note that some variables may be in both of these sets. Since the newly added subset is complete, given V1, that subsystem produces output O1. However, O1 union V2 is an input for the unaugmented system, which, because of its completeness, produces an output showing that the augmented system is complete.

Since the number of subsystems is finite, the process of augmentation ceases after a finite number of steps. By mathematical induction, using a similar argument to that of the previous paragraph, it follows that the entire system is complete.

For KB1, this result can be applied, or alternatively make the following specific argument: Inputs to the system as a whole can be partitioned into inputs for the risk tolerance and the discretionary income subsystems. Each of these is complete, and so produces a risk tolerance and discretionary income respectively. These are inputs to the investment subsystem its only inputs. Since the investment

subsystem is complete it produces an investment. So an output for the entire system exists for each input, and the system as a whole is complete.

Step 4 -- Consistency of the entire system

The first step in proving the consistency of the entire expert system is to prove the consistency of each subsystem. To do this, the user must show that for all possible inputs, the outputs are consistent, i.e., that the AND of the conclusions can be satisfied.

For example, if an expert system concludes "temperature > 0" and "temperature < 100", the AND of these conclusions can be satisfied. However, if the system concludes, "temperature < 0" and "temperature > 100", the AND of these two conclusions has to be false. It is clear that based on the input that produced these two conclusions, it is not possible for all of the system's conclusions to be true at the same time and thus the system producing these conclusions is inconsistent.

Consistency Step 1 -- Find the Mutually Inconsistent Conclusions

The first step in proving consistency is to identify those sets of mutually inconsistent conclusions for each of the subsystems identified in the "Find partitions" step above.

Some sets of conclusions are mathematically inconsistent. For example, if a system describes temperature, the set:

{"temperature < 0", "temperature > 100"}

is mathematically inconsistent.

However, other conclusion sets that are not mathematically inconsistent may be inconsistent based on domain expertise. For example, one investment advisor expert system could be designed to recommend several types of investments to each investor (probably not a bad idea). For such a system, "investment = stocks" AND "investment = bank account" are not inconsistent; stocks and bank accounts are just two of the investments recommended for some investor. However, if the system were designed to recommend only one investment per investor, "investment = stocks" AND "investment = bank account" would be interpreted as a contradiction, and the system recommending this would be inconsistent.

Because some sets of conclusions are inconsistent because of domain expertise, finding all sets of inconsistent conclusions generally requires expert knowledge.

Note that if there are no mutually inconsistent conclusions in the expert system as a whole, then consistency is true by default, and no further consistency proof is necessary.

Consistency Step 2 -- Prove Consistency of Subsystems

If there are inconsistent conclusions in the knowledge base as a whole, then the next step in proving consistency is to prove the subsystems consistent. This can be done by showing that no set of inputs to a subsystem can result in any of the sets of inconsistent conclusions. For each set of inconsistent

conclusions, the user can construct, as detailed below, a Boolean expression B that represents all the conditions under which that set of inconsistent conclusions would be proved by the subsystem. If that Boolean expression can be shown to be FALSE, there are no such conditions.

Now the construction of the Boolean expression B to be proved false will be described. Let

S = {C1, ..., Cn}

be a set of potentially inconsistent conclusions for one of the subsystems.

B will be constructed by a backward chaining process, starting with

B0 = C1 AND ... AND Cn

Let Ci be one of the Cs. For all rules that conclude Ci, construct the OR of these rules initial conditions. Then substitute the resulting expression into B0.

Continue these substitutions until an expression results that has only the inputs to the expert subsystem. For each atomic Boolean expression A that is the conclusion of a rule in the subsystem, substitute the OR of the rule if parts of rules that conclude A. After at most a finite number of such substitutions, the user obtains an expression that states when all the C's would be true in terms of the input variables of the subsystem.

For the risk subsystem, the only inconsistent set of rule conclusions is:

S = { "Risk tolerance" = high and "Risk tolerance" = low }

The only initial conditions for "Risk tolerance" = high is from Rule 3:

"Do you buy lottery tickets" = yes

OR "Do you currently own stocks" = yes

and the only initial conditions for "Risk tolerance" = low is from Rule 4:

"Do you buy lottery tickets" = no

AND "Do you currently own stocks" = no

Let:

A0 = ("Do you buy lottery tickets" = yes)

A1 = ("Do you currently own stocks" = yes).

This means:

not A0 = ("Do you buy lottery tickets" = no)

not A1 = ("Do you currently own stocks" = no).

Using this notation:

B0 = (A0 OR A1) AND (NOT A0 AND NOT A1)

For this small subsystem, B0 is actually expressed in terms of inputs to the subsystem (i.e., B0 is actually B).

Distributing the top level AND over the OR,

B0 = (A0 AND (NOT A0 AND NOT A1))

OR (A1 AND (NOT A0 AND NOT A1))

The first subexpression is FALSE because it contains A0 AND NOT A0. Likewise, the second is FALSE because it contains A1 AND NOT A1. Therefore, B0 is FALSE because it is the OR of only FALSE expressions.

Consistency Step 3 -- Consistency of the Entire System

The results of subsystem consistency are used to establish the consistency of the entire system. The basic argument is to use results on subsystems to prove that successively larger subsystems are consistent. At each stage of the proof, there are some subsystem known to be consistent; initially, this is the subsystem that concludes goals of the expert system as a whole. At each stage of the proof, a subsystem that concludes some of the input variables of the currently-proved-consistent subsystem is added to the currently consistent subsystem. After a number of steps equal to the number of subsystems, the entire system can be shown to be consistent.

When a consistent subsystem that sets input variables of the currently consistent subsystem is added to the currently consistent subsystem, the augmented subsystem is consistent. Any input to the augmented subsystem can be divided into a set V1 of input variables for the unaugmented system and a set V2 for the newly added subsystem. Note that some variables may be in both of these sets. Since the newly added subset is consistent, given V1, that subsystem produces and output O1. However, O1 union V2 is an input for the unaugmented system producing output due to its consistency. This shows that the augmented system is consistent.

Since the number of subsystems is finite the process of augmentation ceases after a finite number of steps. By mathematical induction, using the above mentioned argument, it follows that the entire system is consistent.

For KB1, one can apply the result, or alternatively make the following specific argument: Inputs to the system as a whole can be partitioned into inputs for the risk tolerance and the discretionary income subsystems. Each of these is consistent, and so produces a consistent set of risk tolerance and discretionary incomes, respectively. These are inputs to the investment subsystem, and are that system's only inputs. Since the investment subsystem is consistent, it produces a consistent investment. Thus an output for the entire system exists for each input, and the system as a whole is consistent.

The other subsystems of KB1 can be proved consistent in the same way.

Step 5 -- Specification Satisfaction

In order to prove that KB1 satisfies its specifications, the user must actually know what its specifications are. This is a special case of the general truth that in order to verify and validate, the user must know what a system is supposed to do. Specifications should be defined in the planning stage of an expert system project.

To illustrate the proof of specifications it will be assumed that KB1 is supposed to satisfy:

> A financial advisor should only recommend investments that an investor can afford.

As with many other aspects of verification and validation, expert knowledge must be brought to bear on the proof process. For KB1, an expert might say that anyone can afford a savings account. Therefore, the user only has to look at the conditions under which stocks are recommended. However, that same expert would probably say that just having discretionary income does not mean that the user can afford stocks; that judgment should be made on more than one variable. Therefore, it would be reasonable to conclude that KB1 does not satisfy the above specification.

However, if the expert does agree that the expert system observes all necessary inputs, one must use inputs to the expert system to express a specification. For KB1, this means that the specification is reexpressed as:

> KB1 recommends stocks only when there is discretionary income.

The user can prove this for the investment subsystem by assuming:

> NOT discretionary income

and proving:

> NOT stocks

The only rule that concludes stocks has "discretionary income" = yes in an AND in its "if" part. Therefore, the investment system satisfies the specification.

To prove the entire system satisfies the specifications, the user must look at the conditions under which "discretionary income" = yes is concluded from inputs for the system as a whole. A financial expert would surely say that owning a luxury car or boat does not mean that discretionary income actually exists and the system as a whole fails the specification, an expected outcome of a small example system tackling a complex subject.